The Nature and Evaluation of Commercial Expert System Building Tools

WILLIAM GEVARTER

AI RESEARCH BRANCH, MAIL STOP 244–17
NASA AMES RESEARCH CENTER
MOFFETT FIELD, CA 94035

NASA Ames Research Center
Artificial Intelligence Research Branch

Technical Report FIA–87–05–01–5

May, 1987

The Nature and Evaluation of Commercial Expert System Building Tools

William B. Gevarter
NASA Ames Research Center

ESBTs make it possible to build an expert system in an order of magnitude less time than is possible with Lisp alone. This article reviews such tools.

The development of new expert systems is changing rapidly—in terms of both ease of construction and time required—because of improved expert system building tools (ESBTs.) These tools are the commercialized derivatives of artificial intelligence systems developed by AI researchers at universities and research organizations. It has been reported that these tools make it possible to develop an expert system in an order of magnitude less time than would be required with the use of traditional development languages such as Lisp. In this article, I review the capabilities that make an ESBT such an asset and discuss current tools in terms of their incorporation of these capabilities.

The structure of an expert system building tool

The core of an expert system consists of a knowledge base and an accompanying inference engine that operates on the knowledge base to develop a desired solution or response. If one is to use such a system, an end-user interface or an interface to an array of sensors and effectors is required for communication with the *relevant world.* (A "relevant world" is a system or situation operated on by or in contact with the expert system.) In addition, to facilitate the development of an expert system, an ESBT must also include an interface to the developer

- so that the requisite knowledge base can be built for the particular application domain for which the system is intended,
- so that the appropriate end-user interface can be developed, and
- to incorporate any special instructions to the inference engine (reasoning system) that are required for the particular domain.

The character and quality of these interfaces are two of the main differentiations between commercial tools and ESBTs developed at universities and used in research. Also important in the structure of ESBTs are

- interfaces to other software and databases, and
- the computers on which the ESBTs will run—not only the computers used for development of expert systems, but also those used for their delivery to an end user.

Figure 1 summarizes the structure of an ESBT.

Technical Report List

ARTIFICIAL INTELLIGENCE RESEARCH BRANCH
NASA AMES RESEARCH CENTER
MARCH 1992

FIA

RIA-87-05-01-5

The Nature and Evaluation of Commercial Expert System Building Tools
WILLIAM B. GEVARTER — May 1987

The development of new expert systems is changing rapidly–in terms of both ease of construction and time required–because of improved expert system building tools (ESBTs.) These tools are the commercialized derivatives of artificial intelligence systems developed by AI researchers at universities and research organizations. It has been reported that these tools make it possible to develop an expert system in an order of magnitude less time than would be required with the use of traditional development languages such as Lisp. In this article, I review the capabilities that made an ESBT such an asset and discuss current tools in terms of their incorporation of these capabilities.

RIA-87-07-01-10

Bayesian Prediction for Artificial Intelligence
MATTHEW SELF AND PETER CHEESEMAN — July 1987

This paper shows that the common method used for making predictions under uncertainty in AI and science is in error. This method is to use currently available data to select the best model from a given class of models (this process is called abduction) and then to use this model to make predictions (we call this method transduction). Using transduction, an AI system will not give misleading results when basing predictions on small amounts of data, when no model is clearly best. For common classes of models we show that the optimal solution can be given in closed form.

RIA-87-09-01-0

Introduction to Artificial Intelligence
WILLIAM B. GEVARTER — September 1987

Artificial Intelligence (AI), sometimes referred to as machine intelligence or heuristic programming, is a technology now achieving practicality that has recently attracted considerable publicity. Many applications are now under development and dozens have already been fielded. One simple view of AI is that it is concerned with devising computer programs to make computers smarter. Thus, research in AI is focused on developing computational approaches to intelligent behavior.

The computer programs with which AI is concerned are primarily symbolic processes involving complexity, uncertainty, and ambiguity. These processes are usually those for which algorithmic solutions do not exist and search is required. This, AI deals with the types of problem solving and decision making that humans continually face in dealing with the world.

Knowledge representation

The knowledge that can be easily represented by the tool is a key consideration in choosing an ESBT. As indicated by Figure 2, there are three aspects of knowledge representation that are fundamental to these tools—*object descriptions* (declarative knowledge such as facts), *certainties*, and *actions*. One method of representing objects is by frames with or without *inheritance*. (Inheritance allows knowledge bases to be organized as hierarchical collections of frames that inherit information from frames above them. Thus, an inheritance mechanism provides a form of inference.) *Frames* are tabular data structures for organizing representations of prototypical objects or situations. A frame has slots that are filled with data on objects and relations appropriate to the situation. One version of programming referred to as *object-oriented programming* utilizes objects that incorporate provisions for message passing between objects; attached to these objects are procedures that can be activated by the receipt of messages. Declarative knowledge can also be represented by parameter-value pairs, by use of logic notation, and, to some extent, by rules.

Actions change a situation and/or modify the relevant database. Actions are most commonly represented by rules. These rules may be grouped together in modules (usually as subparts of the problem) for easy maintenance and rapid access. Actions may also be represented in terms of *examples*, which indicate the conclusions or decisions reached. Examples are a particularly desirable form of representation for facilitating knowledge acquisition, and inductive systems capitalize on them. Examples are much easier to elicit from experts than rules, and may often be a natural form of domain knowledge. Actions can also be expressed in logic notation, which is a form of rule representation. Finally, actions can be expressed as procedures elicited by either

- messages (in object-oriented programming) or
- changes in a global database that are observed by *demons*. ("Demons" are procedures that monitor a situation and respond by performing an action when their activating conditions appear.)

In addition to the representation of objects and actions, one must consider the degree to which the knowledge or data is known to be correct. Thus, most ESBTs have provisions for representing certainty. The most common approach is to incorporate "confidence factors"; this approach is a derivative of the approach used in the Mycin expert system.[1] Fuzzy logic and probability are also used.

An alternative way of handling uncertainties or tentative hypotheses is to consider multiple worlds in which different items are true or not true in these alternative worlds. Another consideration is whether or not a *deep model* (which is a structural or causal model) of the system can readily be built with the tool in question as an aid in model-based reasoning. (The same underlying model can often be employed for other uses, such as preservation of knowledge and training.) Finally, system size (for example, as measured by the number of rules needed) can be of critical importance, as it can have an important effect on memory requirements, memory management, and runtimes.

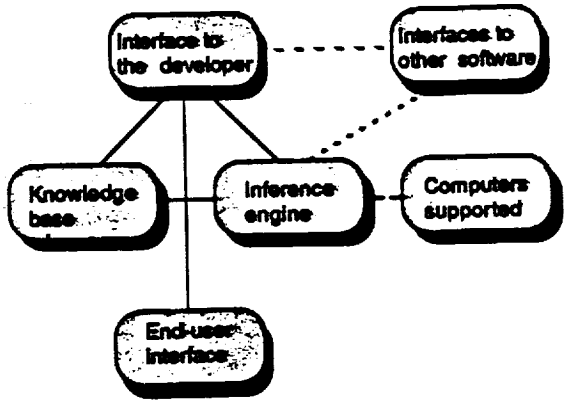

Figure 1. The structure of an expert system building tool. (Solid lines represent basic relationships, and broken lines represent related aspects.)

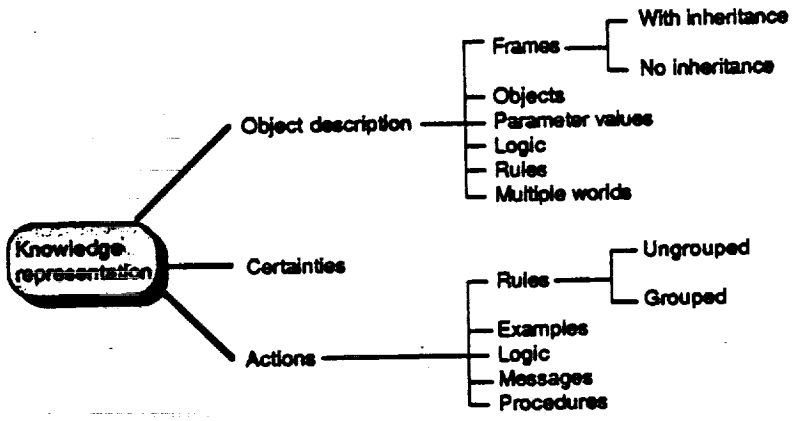

Figure 2. Different methods of knowledge representation.

Inference engine

Figure 3 indicates the major alternative means by which an ESBT performs inferencing. The most usual approach is *classification*, which is appropriate for situations in which there is a fixed number of possible solutions. Hypothesized conclusions from this set are evaluated as to whether they are supported by the evidence. This evaluation is usually done by *backward chaining* through *if-then* (that is, antecedent-consequent) rules, starting with

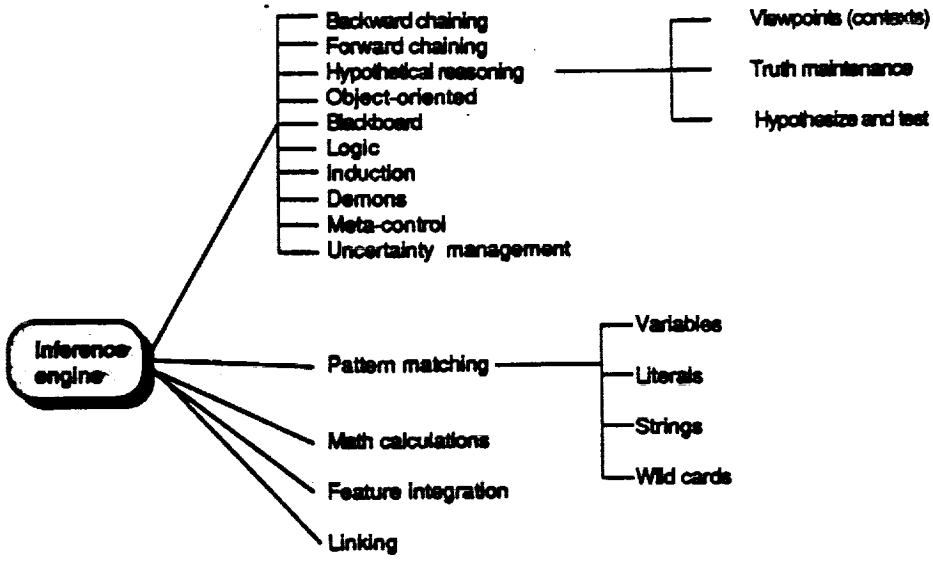

Figure 3. Inference-engine possibilities.

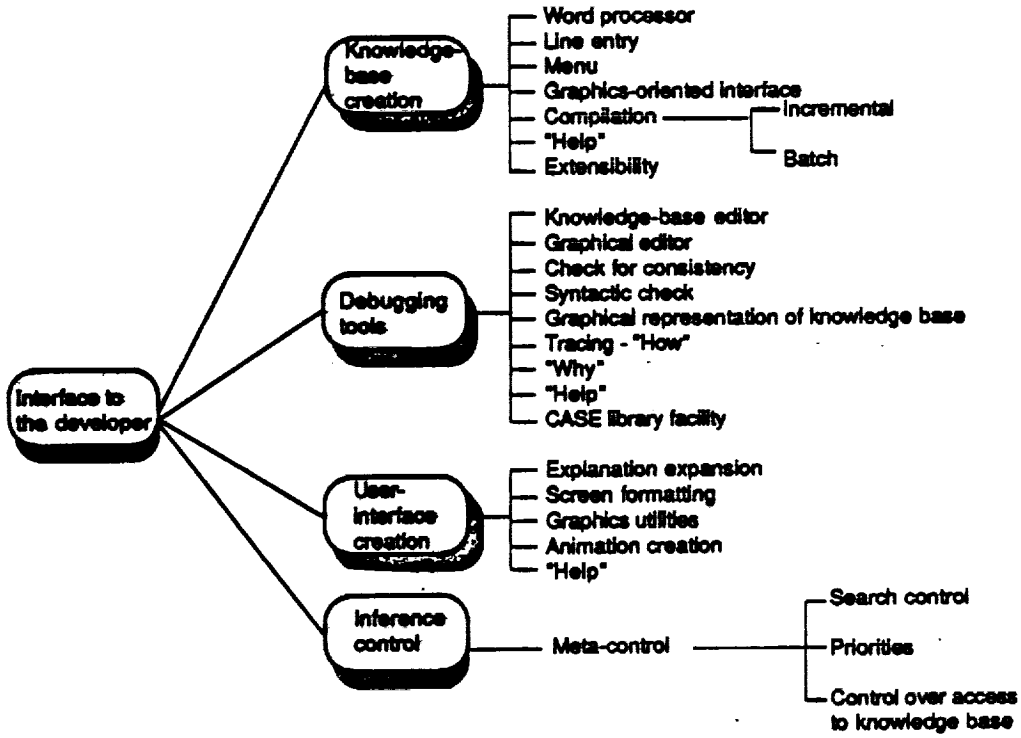

Figure 4. Possibilities for the interface to the developer.

rules that have the hypothesized conclusions as their *consequents*. Rules are then searched for those that have as their consequent the conditions that support the *antecedents* (input conditions) in the hypothesized conclusion rule. This process continues recursively until the hypothesis is fully supported or until either a negation or a dead-end is reached. If either of the latter two events happens, additional hypotheses may be tried until some conclusion is reached or the process is terminated. This depth-first, backward-chaining approach was popularized by the Mycin expert system. The corresponding Emycin ESBT shell[2] is the prototype of virtually all the hypothesis-driven (that is, *goal-driven*) commercial ESBTs currently available.

Forward chaining starts with data to be input or with the situation currently present in a global database. The data or the situation is then matched with the antecedent conditions in each of the relevant rules to determine the applicability of the rule to the current situation. (The current situation is usually represented in the global database by a set of attributes and their associated values.) One of the matching rules is then selected (for example, by the use of *meta-rules*, which help determine the order in which the rules are tried, or by *priorities*), and the rule's consequents are used

2.

to add information to the database or to actuate some procedure that changes the global situation. Forward chaining proceeds recursively (in a manner similar to that of backward chaining), terminating either when a desired result or conclusion is reached or when all relevant rules are exhausted. Combinations of forward and backward chaining have also been found useful in certain situations.

Forward reasoning (a more general form of forward chaining) can be done with data-driven rules or with data-driven procedures (demons).

Hypothetical reasoning refers to solution approaches in which assumptions may have to be made to enable the search procedure to proceed. However, later along the search path, it may be found that certain assumptions are invalid and therefore have to be retracted. This *nonmonotonic reasoning* (that is, reasoning in which facts or conclusions must be retracted in light of new information) can be handled in a variety of ways. One approach that reduces the difficulty of the computation is to carry along multiple solutions (these solutions represent different hypotheses) in parallel and to discard inappropriate ones as evidence that contradicts them is gathered. This approach is referred to as *viewpoints*, *contexts*, and *worlds* in different tools. Another approach is to keep track of the assumptions that support the current search path and to backtrack to the appropriate branch point when the current path is invalidated. This latter approach has been referred to by names like *nonchronological backtracking*. A related capability is *truth maintenance*, which removes derived beliefs when their conditions are no longer valid.

Object-oriented programming is an approach in which both information about an object and the procedures appropriate to that object are grouped together into a data structure such as a frame. These procedures are actuated by messages that are sent to the object from a central controller or another object. This approach is particularly useful for simulations involving a group of distinct objects and for real-time signal processing.

The *blackboard inference approach* is associated with a group of cooperating expert systems that communicate by sharing information on a common data structure that is referred to as a "blackboard." An agenda mechanism can be used to facilitate the control of solution development on the blackboard.

In the case of ESBTs, *logic* commonly refers to a theorem-proving approach involving *unification*. "Unification" refers to substitutions of variables performed in such a way as to make two items match identically. The common logic implementations are versions of a logic-programming language, Prolog, that utilize a relatively exhaustive depth-first search approach.

An important inference approach found in some tools is the ability to generate rules or decision trees inductively from examples. Human experts are often able to articulate their expertise in the form of examples better than they are able to express it in the form of rules. Thus, inductive learning techniques (which are currently limited in their expressiveness) are frequently ideal methods of knowledge acquisition for rapid prototyping when examples can be simply expressed in the form of a conclusion associated with a simple collection of attributes. The human builders of the resultant expert system can then refine it iteratively by critiquing and modifying the results inductively produced. *Inductive inference* usually proceeds by starting with one of the input parameters and searching for a tree featuring the minimum number of decisions needed to reach a conclusion. This *minimum-depth tree* is found by cycling through all parameters as possible initial nodes and using an *information theoretic approach* to select the order of the parameters to be used for the remaining nodes and to determine which parameters are superfluous. An "information theoretic approach" is one that chooses the solution that requires the minimum amount of information to represent it. The depth of the tree is usually relatively shallow (often less than five decisions deep), so large numbers of examples usually result in broad, shallow trees.

Some tools incorporate demons that monitor local values and execute procedures when the actuation conditions of the demons appear. These tools are particularly appropriate for monitoring applications.

A number of tools offer a choice of several possible inference or search procedures. In systems built with such tools, means are usually made available to the system builder to control the choice of the inference strategy, which the builder causes to be dependent on the system state. Such control is referred to as *meta-control*. One form of meta-control is the use of *control blocks*, which are generic procedures that tell the system the next steps to take in a given situation so that the search will be reduced, enabling a large number of rules to be accommodated without the search space becoming combinatorially explosive.

As the certainty of data, rules, and procedures is usually less than 100 percent, most systems incorporate facilities for certainty management. Thus, they have various approaches for combining uncertain rules and information to determine a certainty value for the result.

Pattern matching is often required for mechanizing inference techniques, particularly for matching rule antecedents to the current system state. The sophistication of the pattern-matching approach affects the capabilities of the system. Types of pattern matching vary—from matched identical strings to variables, literals, and wildcards, and can even include partial and/or approximate matching that can serve as analogical reasoning.

Other ESBT capabilities vary from tool to tool. Some inference engines offer rapid and sophisticated math-calculation capabilities. One of the more valuable capabilities is supplied by inference engines that can manage modularized knowledge bases or modularized solution subproblems by accessing and linking these modules as needed.

Another important consideration in a tool is the degree of integration of its various features. Full integration is desirable so that all the tool features can be brought to bear, if needed, on the solution of a single problem. For example, in the case of ESBTs incorporating both object representations and forward and backward chaining rules, it is desirable that expert-system developers be able to mix forward and backward chaining rules freely and be able to reason about information stored in objects when these actions are appropriate.

The interface to the developer

Various tools offer different levels of capabilities for the expert-system builder to use to mold the system. The simpler tools are shells into which knowledge is inserted in a specific, structured fashion. The more sophisticated tools are generally more difficult to learn, but allow the system developer a much wider choice of knowledge base representations, inference strategies, and the form of the end-user interface. Various levels of debugging assistance are also provided. Figure 4 provides an indication of the possible options

3.

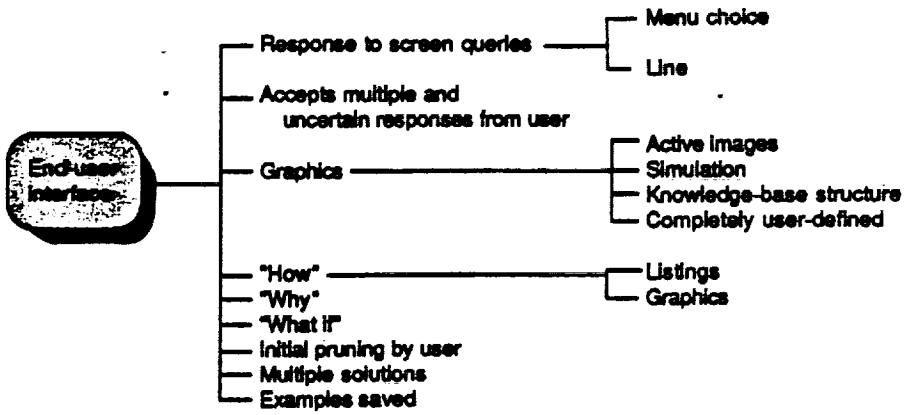

Figure 5. Possibilities for the end-user interface.

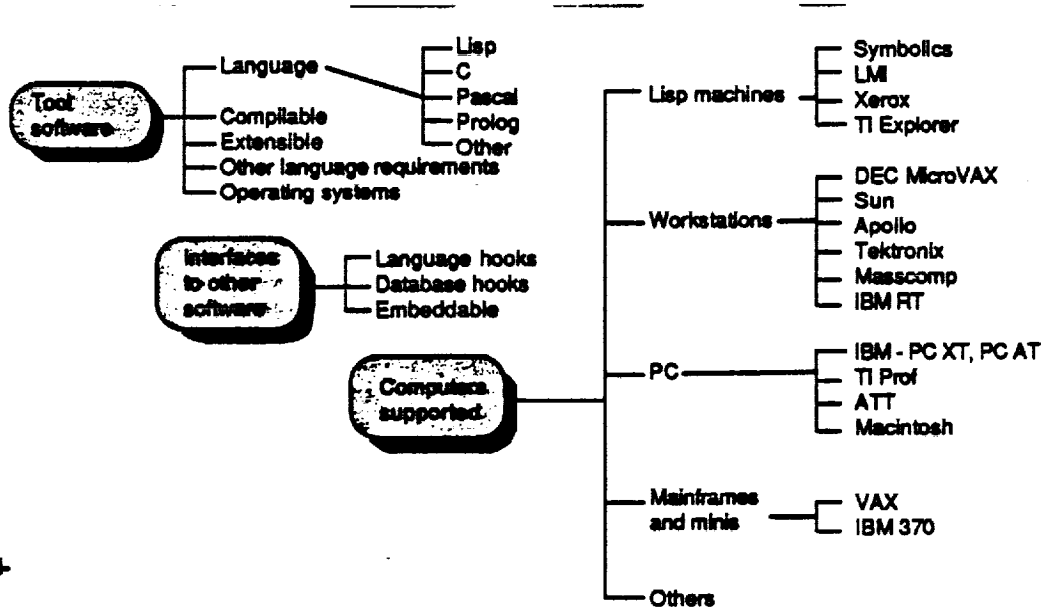

Figure 6. Software and hardware aspects of tools.

(options are tool dependent) that are available for each aspect of the interface to the developer.

End-user interface

Once the expert system has been built, its usability depends in large part on the end-user interface. Figure 5 provides an indication of the range of end-user facilities found in ESBTs. Since most expert systems are really intelligent assistants, the end-user interface is often designed to allow interactive dialogue. This dialogue and/or the initial input most often appear to the user as structured data-input arrangements incorporating menu choices that allow the user to answer requests by the system for information. In some cases, to increase system flexibility, systems will accept multiple and uncertain user responses and still arrive at conclusions (though the certainty of the resultant conclusions is reduced). In sophisticated systems, graphics are often used to show the line of reasoning when the system responds to users' "how" questions; in simpler systems, a listing of the rules supporting the system's conclusions may be employed. ESBTs often answer a user's "Why do you need this information?" question by quoting the rule for which the information is required. The ability of the system to answer the user's "why" and "how" questions is important, for it increases the end user's confidence in the system's decision-making ability.

Other capabilities often found in ESBTs are facilities that allow the end user to select alternative parameter values and observe the effect on the outcome (these facilities support "what if" queries), facilities to allow the user to perform an initial pruning of the line of questioning so that the system need not pursue areas that the user feels are irrelevant or unnecessary, and the capability to save examples for future consideration or use.

Very sophisticated tools often include interactive graphics and simulation facilities that increase the end user's understanding and control of the system being represented. Above all, the end-user interface needs to be user friendly if the system is to be accepted.

Programming-language considerations

In addition to the structure and the paradigms supported by a tool, the programming language in which the tool is

4

written is of major importance. The language determines whether the expert system is compilable and, if it is, whether incrementally or in a batch mode. Compilability reduces the memory requirements and increases the speed of the expert system; incremental compilability speeds development. Figure 6 is illustrative of the aspects related to the tool-language choice.

In general, the more sophisticated tools have been written in Lisp. However, even these tools are now being rewritten in languages such as C to increase speed, reduce memory requirements, and to promote availability on a larger variety of computers. However, some new approaches to mechanizing Lisp may reduce the speed and memory advantages associated with C.

The user can usually extend tools written in Lisp by writing additional Lisp functions. This is also true of some of the other languages, for example, Prolog and Pascal. Similar extensibility is usually found in tools having language hooks for accessing other programs or database hooks for accessing other information. In some cases, the expert system generated by the tool is fully embeddable in other systems, which produces increased autonomy. Whether or not a system is fully embeddable in other systems and is therefore capable of autonomous operations is becoming increasingly important, now that expert systems are moving from prototypes to being fielded. Reliability and memory management (in Lisp, the latter takes the form of *garbage collection*) are often important considerations for fielded systems. "Garbage collection" is the collection of no-longer-used memory allocations; these allocations can slow the system operation.

The computers supported by the various tools are primarily a function of the language and operating system in which the tools are written, and the computer's memory, processing, and graphics-display capabilities. The trend toward making expert system shells available on personal computers (such as those made by IBM) results in part from the increasing capabilities of these computers. However, this trend is also partly owing to the writing of tools in faster languages, such as C, and to taking advantage of modularization in building the knowledge base. As mentioned earlier, such modularization involves decomposing the problem into subproblem modules and providing appropriate linking between these modules as required during operation.

1. Classification
   a. Interpretation of measurements
      Hypothesis selection based on evidence
   b. Diagnosis
      Measurement selection and interpretation
      (often involves models of system organization and behavior)
2. Design and synthesis
   Provide constraints as well as guidance
3. Prediction
   Forecasting
4. Use advisor
   "How to" advice
5. Intelligent assistant
   Provide decision aids
6. Scheduling
   Time-ordering of tasks, given resource constraints
7. Planning
   Many complex choices affect each other
8. Monitoring
   Provide real-time, reliable operation
9. Control
   Process control
10. Information digest
    Situation assessment
11. Discovery
    Generate new relations or concepts
12. Debugging
    Provide corrective action
13. Example-based reasoning
    The source of most rules

Figure 7. AI function capabilities.

Function capabilities

Of primary consideration are the function applications that can readily be built with a particular ESBT. A review of the major function applications follows (see also Figure 7).

Classification. By far the most common function addressed by expert systems is *classification*. "Classification" refers to selecting an answer from a fixed set of alternatives on the basis of information that has been input.

Below are some subcategories of classification.

- *Interpretation of measurements.* This refers to hypothesis selection performed on the basis of measurement data and corollary information.
- *Diagnosis.* In *diagnosis*, the system not only interprets data to determine the difficulty, but also seeks additional data when such data is required to aid its line of reasoning.
- *Debugging, treatment, or repair.* These functions refer to taking actions or recommending measures to correct an adverse situation that has been diagnosed.
- *Use advisor.* An expert system as a front end to a computer program or to a piece of machinery can be very helpful to the inexperienced user. Such systems depend both on the goals of the user and the current situation in suggesting what to do next. Thus, the advice evolves as the state of the world changes. Use advisors can also be helpful in guiding users through procedures in other domains (for example, auto repair and piloting aircraft).

Classification and other function applications can be considered to be of two types: *surface reasoning* and *deep reasoning*. In surface reasoning, no model of the system is employed; the approach taken is to write a collection of rules, each rule asserting that a certain situation warrants a certain response or conclusion. (These situation-response relationships are usually written as heuristic rules garnered from experience.) In deep reasoning, the system draws upon causal or structural models of the domain of interest to help arrive at the conclusion. Thus, systems employing deep models are potentially more capable and may degrade more gracefully than those relying on surface reasoning.

Table 1. A subjective view of the importance of various expert system tool attributes for particular function applications.

● VERY ○ SOMEWHAT LITTLE	DIAGNOSIS AND CLASSIFICATION	DATA ANALYSIS AND INTERPRETATION	DESIGN AND SYNTHESIS	PREDICTION AND SIMULATION	MONITORING	USE ADVISOR	INTELLIGENT ASSIST.	PLANNING AND SCHEDULING	CONTROL
INFERENCE APPROACH									
BC	●	○	○		○	○	○	○	○
FC & FORWARD REASONING	○	●	●	●	●	●	○	●	●
BC, FC, & FORWARD REASON.	●	●	●	○	○	○	●	●	○
HYPOTHETICAL REASONING	●	●	●	○	○	○	○	●	
OBJECT-ORIENTED	○	○	○	●	○	○	○	○	●
BLACKBOARD	●	●	●	○	○	○	○	●	
INDUCTION	●	○	○			○	○		○
OBJECT DESCRIPTION									
FRAMES	●	○	●	○	○	○	○	●	○
FRAMES W/ INHERITANCE	●	○	●	○	○	○	○	●	○
OBJECTS	○	○	○	●	○	○	○	○	●
PARAMETER VALUES PAIRS	○	○	○		○	○			○
LOGIC	○	○	○	○	○	○	○	○	○
RULES	○	○	○	○	○	○	○	○	○
CERTAINTIES	●	●	○	○	○	○	○	○	○
ACTIONS									
RULES	●	●	●	●	●	●	●	●	●
EXAMPLES	●	○	○	○		●	○		○
LOGIC	●	●	●	●	●	○	○	○	○
MESSAGES	○	○	●	●	●	○	○	●	●
PROCEDURES	○	○	●	●	●	○	○	●	●

Design and synthesis. "Design and synthesis" refers to configuring a system on the basis of a set of alternative possibilities. The expert system incorporates constraints that the system must meet as well as guidance for steps the system must take to meet the user's objectives.

Intelligent assistant. Here the emphasis is on having a system that, depending on user needs, can give advice, furnish information, or perform various subtasks.

Prediction. "Prediction" refers to forecasting what will happen in the future on the basis of current information. This forecasting may depend upon experience alone, or it may involve the use of models and formulas. The more dynamic systems may use simulation to aid in the forecasting.

Scheduling. "Scheduling" refers to time-ordering a given set of tasks so that they can be done with the resources available and without interfering with each other.

Planning. "Planning" is the selection of a series of actions from a complex set of alternatives to meet a user's goals. It is more complex than scheduling in that tasks are chosen, not given. In many cases, time and resource constraints do not permit all goals to be met. In these cases, the most desirable outcome is sought.

Monitoring. "Monitoring" refers to observing an ongoing situation for its predicted or intended progress and alerting the user or system if there is a departure from the expected or usual. Typical applications are space flights, industrial processes, patients' conditions, and enemy actions.

Control. Control is a combination of monitoring a system and taking appropriate actions in response to the monitoring to achieve goals. In many cases, such as the operation of vehicles or machines, the tolerable response delay may be as small as milliseconds. In such a case, the system may be referred to as a *real-time system.* "Real-time" is defined as "responding within the permissible delay time" to the end that the system being controlled stays within its operating boundaries.

Digest of information. A system performing this function may take in information and return a new organization or synthesis. One application may be the inductive determination of a decision tree from examples. Others may be the assessment of military or stock market situations on the basis of input data and corollary information.

Discovery. Discovery is similar to digest of information except that the emphasis is on finding new relations, order, or concepts. This is still a research area. Examples include finding new mathematical concepts and elementary laws of physics.

Others. There are other functions, such as learning, that are directly subsumable under the ones I have enumerated thus far. In many cases, these functions (and some of those already mentioned) can be ingeniously decomposed into functions discussed previously. Thus, for example, design and some other functions can often be separated into subtasks that can be solved by classification.

Importance of various ESBT attributes for particular function applications

Table 1 * is an attempt to relate the various attributes that are found in different ESBTs to their importance in facilitating the building of expert systems that perform different functions. A solid circle indicates an attribute that is very worthwhile in helping to build that function. An open circle indicates that it is a lesser contributor. A empty cell indicates an attribute that does not provide a significant contribution. As indicated earlier, the evaluation is subjective because, depending on the insight and ingenuity of the system developer, some of the functions can be decomposed into other functions. Thus, Table 1 reflects what I see as obvious and perhaps necessary attributes for straightforward construction of expert systems that perform the indicated functions.

*In the future, various ESBT approaches may be shown to be Turing Machine equivalents, which would mean that any computation could be performed by them. Therefore, it usually cannot be said definitively that ESBT x cannot perform function y. Thus, Table 1 in the sidebar is really an attempt to reflect my perception of which ESBT attributes simplify the programming of various expert-system functions.

Brief descriptions of commercial ESBTs

The following are descriptions of some of the current commercial expert system building tools in common use. The attributes of these tools are summarized in Tables 1 and 2 for easy comparison. This sidebar is not intended to be an exhaustive survey. For example, VP-Expert, an inexpensive (under $100) but capable rule-based ESBT for PCs, has recently been introduced by Paperback Software in Berkeley, Calif. GEST, an evolving university-supported ESBT from Georgia Tech, provides high-order capabilities (such as multiple knowledge representations) at a fraction of the cost of commercial, more polished tools offering similar capabilities. GURU, from mdbs in Lafayette, Ind., a composite ESBT integrated with a database spreadsheet and natural-language front end, is also available.

ART

ART is a versatile tool that incorporates a sophisticated programming workbench. It runs on advanced computers and workstations such as those produced by Symbolics, LMI, TI, Apollo, and VAX. ART's strong point is *viewpoints*, a technique that allows hypothetical *nonmonotonic reasoning*; in nonmonotonic reasoning, multiple solutions are carried along in parallel until constraints are violated or better solutions are found. At such points, inappropriate solutions are discarded. ART provides graphics-based interfaces for browsing both its viewpoint and *schema* (frame) networks. ART is primarily a forward-chaining system with sophisticated user-defined pattern matching; the pattern matching is based on an enhanced version of an indexing scheme derived from OPS5. (OPS5 is discussed below.) Object-oriented programming is made available by attaching *procedures* (active values) to objects (the objects are called *schemata*). ART has a flexible graphics workbench with which to create graphical interfaces and graphical simulations. ART was designed for near-real-time performance. To achieve this performance, ART compiles its frame-based as well as its relational knowledge into logic-like assertions (the latter are called *discrimination networks*). Applications particularly suited for ART are planning/scheduling, simulation, configuration generation, and design. Currently written in Lisp, ART employs a very efficient, unique memory management system that virtually eliminates garbage collection. A C-language version is now available. (Further information on ART is available from Inference Corp., 5300 W. Century Blvd., Los Angeles, CA 90045; (213) 417-7997.)

KEE

Kee, which runs on advanced AI computers, is the most widely used programming environment for building sophisticated expert systems. Important aspects of KEE are its multifeature development environment and end-user interfaces, which incorporate windows, menus, and graphics. KEE contains a sophisticated frame system that allows the hierarchical modeling of objects and permits multiple forms of inheritance. KEE also offers a variety of reasoning and analysis methods, including object-oriented programming, forward and backward chaining of rules, hypothetical reasoning (which is incorporated as KEE Worlds), a predicate-logic language, and demons. It has an open architecture that supports user-defined inference methods, inheritance roles, logic operators, functions, and graphics. KEE has a large array of graphics-based interfaces that are developer/user controlled, including facilities for graphics-based simulation (the graphics-based simulation facility, Sim Kit, is available at extra cost). KEE has been used for applications in diagnosis, monitoring, real-time process control, planning, design, and simulation. (Further information on KEE is available from IntelliCorp, 1975 El Camino Real West, Mountain View, CA 94040; (415) 965-5500.)

Knowledge Craft

Knowledge Craft (KC) is a hybrid tool based on frames that have user-defined inheritance. It is an integration of the Carnegie Mellon version of OPS5 and of Prolog and the SRL frame-representation language. It is a high-productivity tool kit for experienced knowledge engineers and AI system builders. Frames are used for declarative knowledge; procedural knowledge is implemented by the attaching of demons. KC is capable of hypothetical (nonmonotonic) reasoning when Contexts (a facility offering alternative worlds) is employed. Search is user defined. A graphics-based simulation package (Simulation Craft) is available. Designed to be a real-time system, KC is particularly appropriate for planning/scheduling and, to an extent, is appropriate for process control, but it is something of an overkill where simple classification problems are concerned. (Further information on Knowledge Craft is available from Carnegie Group, Inc., 650 Commerce Court, Station Square, Pittsburgh, PA 15219; (412) 642-6900.)

Picon

Picon is designed as an object-oriented expert system shell for developing real-time, on-line expert systems for industrial automation and other processes that are monitored with sensors during real time; such processes are found in some aerospace and financial applications. Picon operates on the LMI Lambda/Plus Lisp machine and the TI Explorer, which combine the intelligent processing power of a Lisp processor with the high-speed numeric processing and data-acquisition capabilities of an MC68010 processor. The two processors operate simultaneously, enabling Picon to monitor the system in real time, detect events of possible significance in process, diagnose problems, and decide on an appropriate course of action. Picon's icon editor and graphics-oriented display enable a developer with minimal AI training to construct and represent a deep model of the process being automated. Rules about the process are entered by means of a menu-based natural-language interface. Picon supports both forward and backward chaining. (Further information on Picon is available from Lisp Machine, Inc., 6 Tech Dr., Andover, MA 01810; (617) 689-3554.)

S.1

S.1 is a powerful commercial ESBT aimed at structured classification problems. Facts are expressed in a frame representation; judgment-type knowledge is expressed as rules. Though ostensibly a backward-chaining system, S.1 performs forward reasoning by means of a patented *procedural control block technique*. Control blocks can be viewed as implementations of flow diagrams; they guide the system procedure by telling the system the next step to take in the current situation. Control blocks can invoke other control blocks or rules, or they can initiate interactive dialogue. Control blocks are a powerful, knowledge-based means of controlling the search,

Table 2. Attributes of some commercial ESBTs. (This table is continued on page 34.)

NAME	ART 3.0	KEE 3.0	KNOWLEDGE CRAFT	PICON	S.1	ES ENVIRON/ VM OR MVS	ENVISAGE	KES
FUNCTIONAL USES								
Classification	X	X	X	X	X	X	X	X
Design	X	X	X		X			
Planning/Scheduling	X	X	X					
Process Control	X	X	X	X				
KB REPRESENTATION								
Rules								X
Structured Rules	X	X	X	X	X	X	X	
Certainty Factors					X	X	X	X
Rule Limit								
Frames w/Inherit.	X	X	X	X	No inherit.			
Object-Oriented	X	X	X	X				
Logic	X	X	X		X			
Examples								
Structured Examples								
Procedures	X	X	X		X	X	X	X
INFERENCE ENGINE								
Forward Chaining	X	X	X	X	FR	X	X	
Backward Chaining	(goal rules)	X	X	X	X	X	X	X
Demons	X	X	X	X	X		X	
Blackboard-Like	X							
Time-Modeling	X	X		X				
Truth Maintenance	X	X						
Meta Control	X	X	X	X	X	X	X	
Logic	X	X	X		X			
Induction								
Math Calculations	X	X	X	X	X		X	X
Context (Viewpoints, Worlds)	X	X	X					
PATTERN MATCHING								
Variables	X	X	X	X				X
Literals	X	X	X	X	X			
Sequences	X		X	X			X	
Segments	X		X	X				X
Wildcards	X		X					
DEVELOPER INTERFACE								
KB Creation								
Word Processor	X		X		X	X	X	X
* Line Entry		X	X	X	X			X
KB Editor	X	X	X	X	X	X		
Menu	X	X		X	X			
Check for Consistency		X		X	X			
Graphics Rep of KB	X	X	X	X	X			
Inference Tracing	X	X	X	X	X	X	X	X
Graphics Utilities to build end-user interface	X	X	X	X	e			
Screen Format Utilities	X	X	X	X	X	X		
Graphics Simulation Capabilities	X	Sim Kit (Extra cost)	Simulation-Craft (Extra cost)	X	e			
Why	X	X	X		X	X	X	X
How	X	X	X	X	X	X	X	X
Explanation Expansion	X	X	X	X	X	X	X	X
Online Help	X	X	X		X	X	X	
Syntax Help	X	X	X	X	X		X	
Tool Extensible	X	X	X	X			X	

8

	M.1	NEXPERT OBJECT	PERSONAL CONSUL+	EXSYS. 3.0	EXPERT EDGE	ESP FRAME ENGINE	INSIGHT 2+	TIMM	RULE-MASTER	KDS 3	1st CLASS
	X	X X	X	X	X	X X	X	X	X X	X X	X
		X X						X	X	X X	
	X			X							
		Cataloged	X		X	X	X		X		X
	X 1000	2000	X 800	X 5000	X		X 2000	X	X	X 16000 from 4000 examples	X
		X	X			X				X	
						X					X
								X	X	X	X
		X	X			X					
	FR X	X X	X X	X X	X	X X	X X	X	X X	X X	X X
	X	X	X			X				X	X
		X			X					X	
	X	X	X			X X					
	X	X	X	X	X	X	X	X	X	X X	X
	X	X X		Numeric	Numeric	Numeric & String	Numeric		X		Numeric & Logical
				X							
					X						X
	X	X	X	X X	X	X	X	X	X	X	X X
	X X	X X	X X	X X	X X		X	X	X	X X	X X
	X	X X	X	X	X	X X		X	X	X	X X
		X X	X	X	X	X	X	X		X X	X
	X	X X	X	X	X		X	X		X	X e
	X X	X X	X X	X X	X X	X X	X X		X X	X X	TREE
	X X	X X	X X	X X	X X	X X	X X	X	X X	X X	X X
	X	X X	X X		X X	X			X X	X X	X

Table 1: Attributes of some commercial ES/ES. (This table begins on page 32.)

NAME	ART 3.0	KEE 3.0	KNOW-LEDGE CRAFT	PICON	S.1	ES ENVIRON/ VM OR MVS	ENVISAGE	KES
SOFTWARE								
Language of Tool	COMMON-LISP	COMMON-LISP	COMMON-LISP	ZETA-LISP.C	C	PASCAL	PASCAL	C
Compilable	Incremental	Incremental	Incremental	X	X	Incremental	X	X
Other Lang. Req.	COMMON-LISP	COMMON-LISP	COMMON-LISP	LISP		CMS, GDDM, PASCAL/VS VM/CMS or MVS		
Operating Sys.	VMS, UNIX				UNIX VM/CMS VMS		VMS	MSDOS on PC
COMPUTERS SUPPORTED								
Symbolics	X	X	X					
LMI	X	X	X	X				
TI Explorer	X	X	X	X				
VAX	X		X		VMS ULTRIX		X	X
XEROX		X						
IBM PC								X
Macintosh								
TI Prof.								
APOLLO		X			X			X
SUN	X	X			X			X
IBM 370					X	X		
Others		X	X		X		X	X
SYSTEMS INTERFACE								
Lang. Hooks	Via Host Computer	Via Host Computer	Via Host Mach	Via Host Mach	C, Others	X	PASCAL, etc.	C
DB Hooks	Via Host Computer	Via Host Computer	Via Host Mach	Via Host Mach	Via "	X	X	X
END USER INTERFACE								
Screen Capabilities								
Line		X	X	X	X	X	X	X
Menu	X	X	X	X	X	X	X	X
Graphics	X	X	X	X	e			e
Simulation	X	Sim Kit	Simulation-Craft	X	e			
Why	X	X	X	X	X	X	X	X
How	X	X	X	X	X	X	X	X
Help	X	X	X	X	X	X	X	X
Initial Pruning	X	X						
Multiple Solution	X	X	X					X
Examples Saved			X	X	X	X	X	X
Mult. & Uncertain	X	X			X		X	X
User Resp. OK								
What If	X	X	X	X	X	X	X	X
COMPANY	Inference	Intellicorp	Carnegie Group	LMI	Teknowledge	IBM	Sys. Designers Software	S/W Arch. & Engr.
COST (1st unit)	$65K	$52K Includes training	$50K Includes training	$60K	$25K 1 user mach. $45K mult. user	$35K	$25K on VAX $15K on MicVAX	$4K PC $7K Wk Stns $25K VAX
Run Time Sys. Cost	$1-5K	PC Host + VAX $20K + 0.5K/PC	None Yet		$9.5K	$25K		10% of Develop Sys.

	M.1	NEXPERT OBJECT	PERSONAL CONSUL+	EXSYS. 3.0	EXPERT EDGE	ESP FRAME ENGINE	INSIGHT 2+	TIMM	RULE-MASTER	KDS 3	1st CLASS
	C X	C Incremental	LISP X	C Incremental	C X	Prolog 2 Incremental	Turbo-PASCAL X	FORTRAN X	C X	8086 Assembly	PASCAL X
	MSDOS	MSDOS VMS	MSDOS	MSDOS VMS	MSDOS	MSDOS	MSDOS		C Compiler DOS 3.0 UNIX, VMS	MSDOS	MSDOS
				VMS UNIX				X	X		
	X	AT	X	X	X	X	X	X	AT	X	X
		X	X					X			X
									X X		
		X					X	X	X		
	X X	C, PASCAL DBASE III	LISP DBASE III	X X	X X	Prolog 2 Via "	y DBASE II	X	Most X	PASCAL, BASIC Reads Assem. Lang.	ANY X
	X	X	X X	X	X X	X X	X	X	X	X	X
		X X	e	e						X X	e
	X X	X X	X X	X X	X X	X X	X X	X	X X	X X	X
	X	X	X X	X	X X	X	X X		X	X X	X
		X X	X X	X X	X X	X		X		X X	X
	X	X	X	X	X		X	X		X	
	Teknow-ledge	Neuron Data	TI	Exsys, CA Intell	Human Edge	Exp Sys Inter	Level-5-R	GRC	Radian	KDS Dev Sys	Progr. in Motion
	$5K	$5K PC-AT $3K MAC	$3K	$395 PC $5K VAX	$2500 Adv $5000 Pro	$895	$485	$1.9K PC $19K others	$995 PC $5K Wkst $17.5K VAX	$1495	$495
	- $50	$1K	$75 First Unit	One Time Fee	$50	By Agreement	Negoti.	No Fee	$100 PC $500 UNIX/VMS	Based on Quality	No Fee

and thus they have made it possible for one to write programs containing thousands of rules without being overwhelmed by a combinatorial explosion (runtimes tend to be linear with the number of rules). S.1 is written in C and executes very rapidly. A major advantage of S.1 is that it can readily be integrated into existing software. A delivery version is available without the system-development portion of S.1; the delivery version can be completely embedded in applications. S.1 is not aimed at exploratory programming; it is aimed at commercial applications in which iterative development of solutions to solvable problems is desired. S.1 has an excellent user interface that features mouse-driven, graphical representations of both the knowledge bases and the inference traces. Problems can be solved in terms of subproblems, which can be linked to handle the complete problem (consistency checking is performed as part of linkage). All S.1 features are expressed in an integrated, strongly typed, block-structured language that facilitates system development and long-term maintenance. (Further information on S.1 is available from Teknowledge, Inc., 1850 Embarcadero Rd., PO Box 10119, Palo Alto, CA 94303; (415) 424-0500.)

Table 2. A composite view of some commercial ESBTs.

SYSTEM	KNOW. REP.						INFERENCE ENGINE						FUNCTIONAL CAPABIL.							DEV. INTERFACE			END USER INTER.	
● STRONG ○ FAIR □ USER PROG.	OBJ. REP.	CERTAINTIES	ACTIONS	EXAMPLES	PATTERN MATCH	BC	FC	OBJ.-ORIENTED	LOGIC	INDUCTION	HYPO. REASON.	CLASS. INTERPRETA. ADVICE (SHALLOW)	CLASS. INTERPRETA. ADVICE (DEEP)	DESIGN	PLANNING	MONITOR/CONTROL	KB CREATION	DEBUGGING	GRAPHICS	USER INT. CREA.	INFER. CONTROL	WHY/HOW	SCREEN INTER.	GRAPHICS	WHAT IF
ART	●	□	●		●	●	●	●	●		●	●	●	●	●	○	●	●	●	●	○	●	●	●	●
IKEE	●	□	●		●	●	○	●	●		●	●	●	●	●	○	●	●	●	●	●	●	●	●	●
KNOWLEDGE CRAFT	●	□	●		●	●	●	●	●		●	●	●	●	●	○	●	●	●	●	□	●	●	●	○
PICON	●		●		●	●	●	●				●	●			●	●	●	●	●	○	●	●	●	●
S. 1	○	C	●		○	●	FR		○			●	●	○			●	●	●	●	●	●	●	○	●
ES ENVIRON/VM OR VMS		C, F	●		○	●	○					●				○	●	○		●	○	●	○		●
ENVISAGE (ADVISOR)	○	F, S	●		○	●	○					●	○				○	●		○	○	●	○		○
KES		C, S	●		○	●	○				●	●	○				○	○		○		●	○		●
M. 1		C	●		○	●	FR		○			●	○	○	○	○	●	●		●	○	●	○		
NEXPERT OBJECT	●	□	●		○	●	●					●	○	○	○	○	●	●	●	●	●	●	●	●	●
PERSONAL CONSULTANT +	○	C	●			●	○					●	○				●	●	○	●	●	●	●	○	●
EXSYS 3.0		C	○		○	●	○					○					●	●		○		●	●	○	●
EXPERT EDGE		S	○		○	●						●					●	●		●		●	●		○
ESP FRAME ENGINE	●		●		○	●	○		●			●	●				●	●	○	○	○	●	●		
INSIGHT 2 +		C	○		○	●	○					●					●	●		○		●	○		●
TIMM		C, I	○	●	●		●				●	●	○			○	●	●		○		○	●	●	○
RULEMASTER 3.0		F	○	●	○	●	○				●	●		○		○	●	●		○		○	○		
KDS 3	○	C		●		●	●	○		●		●	○	○	○	○	●	○	○	●	○	○	●	●	○
1st CLASS		C	○	●	○	○	○			●		●					●	●	○	○		○	○		○
OPS5+			●		●		●					●	○	○	○	○	●	●			○		○		

FW — FORWARD REASONING
C — CONFID. FACTOR
F — FUZZY LOGIC
B — BAYESIAN LOGIC
I — INCOMPLETE INFO
L — LISP
PA — PASCAL
A — ASSEMBLY
F — FORTRAN
PR — PROLOG
O — OTHERS
C — C
D — DISCRIM. NET
R — RULES
IN — INCREMENTAL
T — TMS
DT — DECIS. TREE

ES Environment/VM or MVS (ESE/VM or ESE/MVS)

ESE is an improved version of Emycin; it is designed for classification problems, but does allow for forward chaining. It consists of two components: a development interface and a consultation interface. A Focus Control Block mechanism has been added to allow the developer to modify and control the flow of inference and, thus, to increase the system speed. ESE/VM and ESE/MVS have good utilities for enabling the developer to fashion the user interface and to incorporate graphics in the user interface when appropriate. ESE is particularly suitable for IBM mainframe users who must interface with existing software and databases. (Further information on ESE is available from IBM, Dept. M52, 2800 Sand Hill Rd., Menlo Park, CA 94025; (415) 858-3000.)

	S/W H/W CHARAC.				
	▪ TOOL LANG.	● COMPILABLE	LANG/DB HOOKS	COMPUTERS ●●	$ 1st UNIT COST
	L	D, IN	O	V, Su, S, L, T	65
	L	IN	O	A, S, L, T, X, O	52† + SMI KIT
	L	IN	O	L, S, T, V, O	90† + SMI
	L, C	Y	●	L, T	60
	C	Y	●	370, V, Su, H, O	25, 45
	PA	IN	●	370	60
	PA	Y	●	V, μV	25, 15
	C	Y	O	I, A, V, O, Su, μV	4, 7†, 25†
	C	Y	●	I	5
	C	IN	●	I, M, O	3, 5
	L	Y	●	I, TP, T	3
	C	IN	O	I, V	8.4, 5
	C	Y	O	I	2.5, 5
	PR	IN	O	I	0.9
	PA	Y	●	I, O	8.5
	F	R	●	TP, I, V, O	1.9, 19
	C	DT	●	I, A, Su, V	1, 5, 17.5
	A	R	●	I, TP	1.5
	PA	DT	●	I, TP	8.5
	C	Y	O	A, Su, I, M, O	1.8, 3

●● 370 — IBM 370 · IM — MACINTOSH · TP — TI PC · I — IBM PCs · S — SYMBOLICS · L — LMI · T — TI EXPLORER · X — XEROX · V — VAX · μV — MICROVAX · Su — SUN · A — APOLLO · O — OTHERS

† — INCLUDES TRAINING

Envisage

Envisage is a Prolog-derived tool. Thus, instead of entering rules, one enters logical assertions. Non-Prolog features include demons, fuzzy logic, and Bayesian probabilities. Envisage is primarily aimed at classification problems. (Further information on Envisage is available from System Designers Software, Inc., 444 Washington St., Suite 407, Woburn, MA 01801; (617) 935-8009.)

KES

KES is a three-paradigm system that supports production rules, hypothesize-and-test rules (hypothesize-and-test rules use the criterion of minimum set coverage to account for data), and Bayesian-type rules for domains in which knowledge can be represented probabilistically. KES is primarily geared to classification-type problems. KES can be embedded in other systems. The hypothesize-and-test approach starts with a knowledge base of diagnostic *conclusions* (that is, classifications) with their accompanying *symptoms* (also called "characteristics"). The session begins with the selection by the system of the set of all diagnoses that match the first symptom of the given problem; the system then reduces this set as the remaining problem symptoms are considered. If the initial set of diagnoses does not include all the remaining symptoms, new diagnoses are added to the set to cover these cases. (Further information on KES is available from Software Architecture and Engineering, Inc., 1600 Wilson Blvd., Suite 500, Arlington, VA 22209-2403; (703) 276-7910.)

M.1

M.1 is a PC-based ESBT targeted for solvable problems rather than for exploratory programming. It is a basically a backward-chaining system designed for classification. It includes the capability for meta-level commands that direct forward reasoning. Written in C, it can readily be integrated into existing conventional software. Its main drawback is that it has no true object-description capability and therefore cannot readily support deep systems. However, M.1 does have a good set of development tools and developer- and user-friendly interfaces. (Further information on M.1 is available from Teknowledge, Inc., 1850 Embarcadero Rd., PO Box 10119, Palo Alto, CA 94303; (415) 424-0500.)

Nexpert Object

Nexpert Object is a powerful, rule-based tool coded in C to run on a Macintosh with 512K of RAM, the Mac Plus, or the IBM PC AT. It has editing facilities comparable with those found on a large tool designed to run on the more sophisticated AI machines. The system allows the developer to group rules into categories so that the rules need be called up only when they are appropriate. Nexpert Object supports variable rules and combinations of forward and backward chaining. The system can automatically generate graphical representations of networks of rules; these representations of networks indicate how the rules relate to one another. Similar networks can be generated to show rule firings that take place in

response to a particular consultation. Nexpert Object includes the capabilities of both frame representations that have multiple inheritance and of pattern-matching rules, so deep reasoning is facilitated. Nexpert Object is a sophisticated system with a focus on the graphical representation of both the knowledge bases and the reasoning process, which makes possible natural and comprehensible interfaces for both the developer and end user. (Further information on Nexpert Object is available from Neuron Data Corp., 444 High St., Palo Alto, CA 94301; (415) 321-4488.)

Personal Consultant+ (PC+)

PC+ is an attempt to provide on a personal computer many of the advanced features found in more sophisticated tools; such tools include KEE. Thus, PC+ utilizes frames with attribute inheritance, and rules. PC+ supports the backward-chaining approach derived from Emycin. It also includes forward-chaining capabilities without variable bindings. PC+ has an extensive set of tools for both development and execution that incorporate user-friendly interfaces. The new 2.0 version supports up to 2M bytes of expanded or extended memory for increased knowledge-base capacity. It also supports the IBM Enhanced Graphics Adapter and access to the popular dBase II and III database packages on the IBM PC. A version of PC+ is also available for the TI Explorer Lisp Machine. PC Easy, a simplified version of PC+ without frames, is also offered. (Further information on PC+ is available from Texas Instruments, Inc., PO Box 209, MS 2151, Austin, TX 78769; (800) 527-3500.)

Exsys 3.0

Exsys 3.0 is written in C for PCs as an inexpensive, rule-based, backward-chaining system oriented toward classification-type problems. Rules are of the if-then-else type. Exsys includes a runtime module and a report generator. Exsys can interface to the California Intelligence company's after-market products: Frame (to provide frame-based knowledge representation) and Tablet (to provide a blackboard knowledge-sharing facility that incorporates tables). (Further information on Exsys 3.0 is available from Exsys, Inc., PO Box 75158, Contr. Sta. 14, Albuquerque, NM 87194; (505) 836-8676.)

Expert Edge

Expert Edge is basically a rule-based, backward-chaining system aimed at rapidly prototyping and delivering classification applications in the 50-to-500 rule range. It uses probabilities and Bayesian statistics to handle uncertainties and lack of information. Its outstanding features are its excellent developer and end-user interfaces, which feature pop-up windowing environments. These are accompanied by a natural-language interface and very good debugging facilities. The professional version interfaces with a video disk and is also able to do extended mathematical calculations. (Further information on Expert Edge is available from Human Edge Software Corp., 1875 S. Grant St., San Mateo, CA 94402-2669; (415) 573-1593.)

ESP Advisor and ESP Frame-Engine

ESP is a Prolog-based system that is particularly appropriate for designing expert systems that guide an end user in performing a detailed operation involving technical skill and knowledge. The developer builds the system by programming in KRL (Knowledge Representation Language), a sophisticated and versatile language that supports numeric and string variables, including facts, numbers, categories, and phrases. Prolog's heritage is clearly apparent in the system's ability to support a full set of logic operators, which enables the developer to write efficient, complex rules. The ESP consultation shell offers a well-designed, multipanel display that makes good use of color. A *text-animation* feature allows the developer to insert text at any point in a consultation. Though ESP Advisor was designed as an introductory prototype tool, its extensibility makes expert systems of greater complexity possible. ESP Frame-Engine supports frames with inheritance, forward and backward chaining rules, and demons. (Further information on the ESP products is available from Expert Systems International, 1700 Walnut St., Philadelphia, PA 19103; (215) 735-8510.)

Insight 2+

Insight 2+ is primarily a rule-based, backward-chaining (that is, goal-driven) system, but it can support forward chaining as well. Facts are represented as elementary objects with single-value or multivalue attributes. Rules are entered in PRL (Production Rule Language). The knowledge base is compiled prior to runtime. Uncertainty is handled by means of confidence factors and thresholds. Because Insight 2+ lacks methods for representing deep models, it is best used for heuristic problems, for which it is a useful tool. Its ability to access external programs and databases is a major enhancement. (Further information on Insight 2+ is available from Level Five Research, Inc., 503 Fifth Ave., Indialantic, FL 32903; (305) 729-9046.)

TIMM

TIMM is an inductive system that builds rules from examples. Examples are first translated into rules, which are then used to build more powerful generalized rules. TIMM handles contradictory examples by arriving at a certainty that is based on averaging these examples' conclusions. Partial-match analogical inferencing is used to deal with incomplete or nonmatching data. TIMM indicates the reliability of its results. The expert systems that result from it can be embedded in other software programs. (Further information on TIMM is available from General Research Corp., 7655 Old Springhouse Rd., McLean, VA 22102; (703) 893-5900.)

Rulemaster 3.0

Though Rulemaster is capable of independent forward and backward chaining, its major distinguishing feature is its capability for inductively generating rules from examples. It also offers fuzzy logic. Interaction with the knowledge base is accomplished by means of a text editor. If they prefer, knowledge engineers can develop Rulemaster applications by writing code directly in the high-level Radial language of Rulemaster instead of using examples. However, a strong programming

14

background is required for easy usage. Rulemaster can generate C or Fortran source code for fast execution, compactness, and for creation of portable expert systems that can interface to other computer programs. (Further information on Rulemaster is available from Radian Corp., 8501 Mo-Pac Blvd., PO Box 9948, Austin, TX 78766; (512) 454-4797.)

KDS3

KDS3 inductively generates rules from examples. Examples can be grouped to develop knowledge modules, which KDS calls *frames* and which can be chained together to form very large systems. Both forward and backward chaining are supported. KDS3 can take input from external programs and sensors and can drive external programs. Expert systems built with KDS3 can be made either (a) interactive or (b) fully automatic for intelligent process control. The entire system is written in assembly language for very rapid execution on PCs. Graphics can be incorporated automatically from picture files or, if one makes use of built-in KDS3 color graphics primitives, they can be drawn in real time. KDS3 incorporates a blackboard by means of which knowledge modules can communicate. KDS2 without the blackboard facility is also available. (Further information on KDS3 is available from KDS Corp., 934 Hunter Rd., Wilmette, IL 60091; (312) 251-2821.)

1st-Class

This is an induction system that generates decision trees, which are elaborate rules, from examples given in spreadsheet form. Problems can be broken down into modules derived from sets of examples; the modules can be chained together with forward or backward chaining. Rules can also be individually built or edited in graphical form on the screen. Several algorithms are available for inferencing: The system can match queries to examples that exist in the database, or the system can utilize the rule trees either as generated or in the *preferred mode*, which employs optimized rule trees that ask questions in the best order. Because all the rules are compiled, the system is very fast. The 1st-Class induction system is designed to interface readily with other software. (Further information on 1st-Class is available from Programs in Motion, Inc., 10 Sycamore Rd., Wayland, MA 01778; (617) 653-5093.)

OPS5

Various versions of the OPS5 expert-system-development language, developed at Carnegie Mellon University, are available. OPS5 is a forward-chaining, production-rule tool with which many famous expert systems used at DEC, such as R1/XCON, have been built. OPS5 pattern-matching language permits variable bindings. However, OPS5 does not have facilities for sophisticated object representations. In general, the development environment is unsophisticated, although some debugging-and-tracing capability is usually provided. The use of a sophisticated indexing scheme (the Rete algorithm) for finding rules that match the current database makes OPS5 one of the tools that executes the fastest. Unfortunately, it is not an easy tool for the nonprogrammer to use. Variations of a representative version, OPS5+, can be obtained for the IBM PC, Macintosh, and the Apollo Workstation. (Further information on OPS5 is available from Computer Thought Corp., 1721 West Plano Pkwy., Suite 125, Plano, TX 75075; (214) 424-3511.)

Attributes of particular commercial ESBTs

The sidebar entitled "Brief descriptions of commercial ESBTs" presents some of the better-known commercial ESBTs. Attributes of these ESBTs are listed in Table 1 of that sidebar. Inclusion of an ESBT in the sidebar in no way represents an endorsement of that product. The descriptions and listings have been constructed from company and noncompany literature, discussions with company representatives, demonstrations, exploratory use of the tools, and so on. However, some incompleteness, errors, and oversights are inevitable in such an endeavor, so it behooves the interested person to use this material as a guide and to examine the systems directly. Direct examination is particularly important because increasing competition is forcing ESBT developers to make rapid improvements and changes in both their systems and their prices.

A comparative, composite view of the various tools

Table 2 of the sidebar provides a composite view of the various ESBTs. Many of the attributes have been integrated to provide a more easily understandable picture of the capability of the tools in each subcategory (for example, the rule and procedure attributes have been combined into "representation of actions"). A solid circle indicates that the tool appears to be strong in a subcategory, an open circle indicates that it appears to be fair, and an empty cell indicates little or no capability in that area. Note that by relating each tool's attributes to its functional importance, I have attempted to indicate each tool's suitability for developing various function applications. Also, note that the more expensive (and correspondingly more sophisticated) tools have the widest applicability. This is often because they are a collection of different paradigms incorporated into a single tool. As a result, they may often be regarded as higher order programming languages and environments, instead of as simple shells into which information is inserted to create an expert system directly. The shell model is more nearly true of the simpler induction systems; such systems can be considered as knowledge-acquisition and rapid-prototyping tools from which more com-

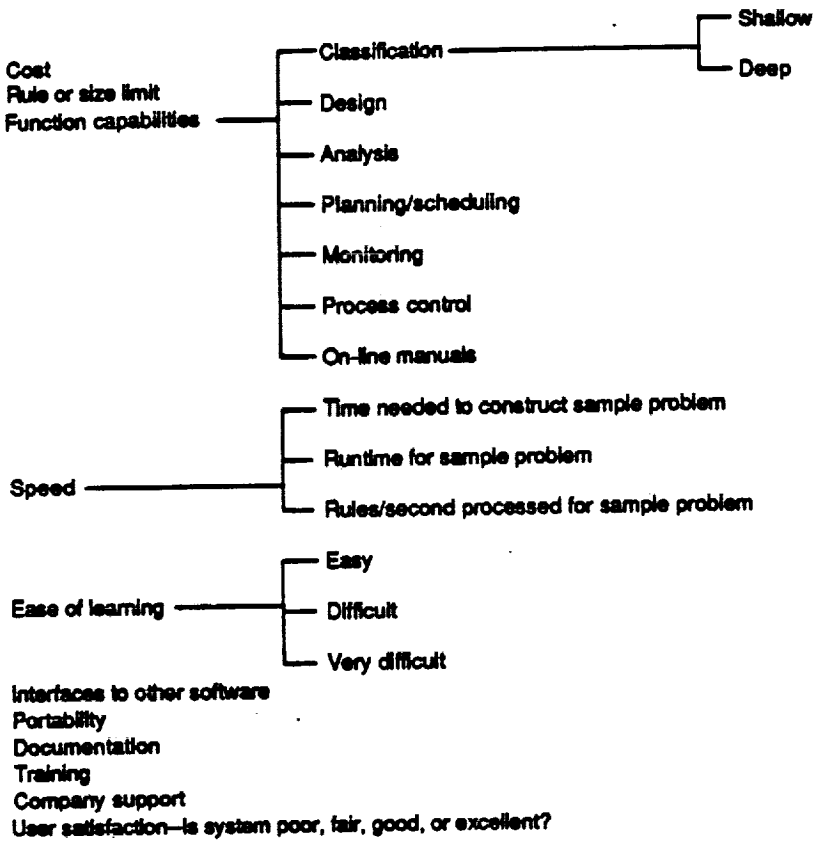

Figure 8. Considerations for assessing the overall usability of a tool.

plex systems can be built by means of other tools by enlarging upon the simple rules inductively generated.

Overall usability of a tool

Figure 8 summarizes some of the aspects that enter into the critical ESBT attribute "overall usability of a particular tool." In addition to obvious factors such as costs and function applicability (function applicability is a measure of which functions are easily accomplished with a tool and which are difficult to accomplish with it), tool choices should be guided by the size of the system to be built, how rapidly a system of the given size and complexity can be built with the tool, and the speed of operation of the tool both during development and, particularly, during end use. (During end use, sub-elements of the tool act as a software delivery vehicle for the developed expert system.) Perhaps the most important factor, however, is the degree of satisfaction of both the developer and the end user. This is related to how obvious the uses of the tool features are, how direct the lines of action to the user's or developer's goals are, the control the developer and end user sense that they have over the system, the nature of the interaction or display (for example, whether they take place by means of menu or graphics), how easy it is to recover from errors, the on-line help that is furnished, and the perceived esthetics, reasonableness, and transparency of the system. Also of major importance is how easy it is to learn the system. This often depends on many of the factors already discussed, but is also closely related to how apparent the choice is at each step (for example, the apparent choice when menus are used is different from the apparent choice when programming is required), the quality of the documentation and on-line help, and the ESBT's structure. Manufacturer-sponsored courses help; however, these are often expensive and inconvenient. A related factor is manufacturer support of the tool, particularly the availability of help over the telephone when it is required.

Finally, such factors as the system's portability, the computers it will run on, the delivery environment, the system's capability of interfacing with other programs and databases, and whether the developed system can be readily embedded in a larger system are all important in an evaluation of a tool. A more difficult factor to evaluate is the ease of prototyping versus life cycle cost. As prototypes are expanded into fielded systems and as they are iteratively further expanded and updated, difficulties are often encountered in system stability, runtime, and memory management.

Though many of these factors can be deduced from the tool's specifications and from system demonstrations, in many cases one can properly differentiate between two tools intended for the same application only if he or she learns both systems and attempts to build the same set of applications with each one. Nevertheless, the factors described in this article and the initial evaluation furnished in the sidebar should prove useful as initial guides to potential users.

To date, ESBTs have made possible productivity improvements of an order of magnitude or more in constructing expert systems. Current tools are only forerunners of ESBTs yet to come. The trend is toward less expensive, faster, more versatile, and more portable tools that will readily make possible development of expert systems that can directly communicate with existing conventional software such as databases and spreadsheets, and can also be embedded into larger systems, with resulting autonomous operations. Higher-end ESBTs are now moving from Lisp machines to more conventional workstations that are less expensive. Lower-end systems are becoming more capable and now appear on IBM PCs and Macintoshes. Delivery systems, which utilize a subset of the complete ESBTs (ESBTs with the development portion removed) are now allowing the completed expert system to be delivered on personal computers or workstations. In addition, versatility will be enhanced with increased choices of inference engines such as blackboard systems. Also in the works are modular ESBTs that will allow the developer to choose various knowledge representations and inference techniques as he or she desires and still be able to build an integrated system. Already appearing are ESBTs coupled to other software sys-

16.

tems such as databases and spreadsheets. Also beginning to appear are expert systems that are specialized to specific functions such as scheduling, process control, and diagnosis.

Finally, the developer and end-user interfaces are getting friendlier and more capable. One of the things providing greater capability is the increased use of graphics and graphical simulations. It is expected that as these friendlier systems emerge, there will be increased development of expert systems directly by the experts themselves.

The rich and growing variety of ESBTs may make it more difficult to choose a tool, but if it is properly selected, the tool will be more closely matched to developer and end-user needs. □

References

1. E.H. Shortliffe, *Computer-Based Medical Consultations: Mycin*, Elsevier-North Holland, New York, 1976.
2. W. van Melle, "A Domain-Independent System that Aids in Constructing Knowledge-Based Consultation Programs," tech. report No. 820, 1980, Computer Science Dept., Stanford University, Stanford, Calif.

Suggested reading

Gevarter, W.B., *Intelligent Machines*, Prentice-Hall, Englewood Cliffs, N.J., 1985.

Gilmore, J.F., and K. Pulaski, "A Survey of Expert System Tools," *Proc. Second Conf. on Artificial Intelligence Applications*, Dec. 11-13, 1985, Computer Society Press, Los Alamitos, Calif., pp. 498-502.

Gilmore, J.F., K. Pulaski, and C. Howard, "A Comprehensive Evaluation of Expert System Tools," *Proc. SPIE Applications of Artificial Intelligence*, Apr. 1986, SPIE, Bellingham, Wash.

Harmon, P., and C.D. King, *Artificial Intelligence in Business*, John Wiley & Sons, New York, 1985.

Hayes-Roth, F., D.A. Waterman, and D.B. Lenat, *Building Expert Systems*, Addison-Wesley, Reading, Mass., 1983.

Karna, A., and A. Karna, "Evaluating Existing Tools for Developing Expert Systems in PC Environment," *Proc. Expert Systems in Government*, K.N. Karna, ed., Oct. 24-25, 1985, Computer Society Press, Los Alamitos, Calif., pp. 295-300.

Riley, G.D., "Timing Tests of Expert System Building Tools," NASA JSC Memorandum FM 7 (86-51), Apr. 3, 1986, FM7/Artificial Intelligence Section, Johnson Space Center, Houston, Tex.

Richer, M.H., "An Evaluation of Expert System Development Tools," *Expert Systems*, Vol. 3, No. 3, July 1986, pp. 166-183.

Waterman, D.A., *A Guide to Expert Systems*, 1986, Addison-Wesley, Reading, Mass., pp. 336-379.

"AI Development on the PC: A Review of Expert System Tools," *The Spang Robinson Report*, Vol. 1, No. 1, Nov. 1985, pp. 7-14.

"Expert Systems-Building Tools," *Expert Systems Strategies*, P. Harmon, ed., Vol. 2, No. 8, Aug. 1986, Cutter Information, Arlington, Mass., pp. 17-24.

"Small Expert Systems Building Tools," *Expert Systems Strategies*, P. Harmon, ed., Vol. 1, No. 1, Sept. 1985, Cahners Publishing, Newton, Mass., pp. 1-10.

Catalogue of Artificial Intelligence Tools, A. Bundy, ed., Springer-Verlag, New York, 1985.

PC (issue on expert systems), Vol. 4, No. 8, Apr. 16, 1985, pp. 108-189.

William B. Gevarter is a computer scientist with the Artificial Intelligence Research Branch at NASA Ames Research Center. Before being transferred to NASA Ames in 1984, he spent several years at the National Bureau of Standards on special assignment from NASA writing a series of NASA/NBS overview reports on artificial intelligence and robotics. Prior to that assignment, he was manager of automation research at NASA headquarters. His current interests are expert systems and human and machine intelligence.

Gevarter received his BS (1951) in aeronautical engineering from the University of Michigan, his MS (1955) in electrical engineering from the University of California at Los Angeles, and his PhD (1966) in aeronautics and astronautics with a specialization in optimal control from Stanford University.

Readers may write to William Gevarter at NASA Ames Research Center, MS 244-17, Moffett Field, CA 94035.

CALL FOR PARTICIPATION

The Fourth IEEE Conference on

ARTIFICIAL INTELLIGENCE APPLICATIONS

Sheraton Harbor Island Hotel
San Diego, California
March 14-18, 1988

sponsored by: The Computer Society of the IEEE

This conference is devoted to the application of artificial intelligence techniques to real-world problems. Only new, significant and previously unpublished work will be accepted. Two kinds of submissions are appropriate:

Those that focus on AI techniques or implementations and show how they can be applied effectively to important problems; and

Those that focus on particular AI-based application programs that solve significant problems, including an analysis of why the implementation techniques selected are appropriate for the problem domain.

AI techniques include: Knowledge representation, reasoning, uncertainty, knowledge acquisition, learning, natural language, intelligent interfaces, and general tools.

Application areas include: Science, medicine, law, business, engineering, manufacturing, and robotics

The following types of submissions will be reviewed by the Program Committee:

Full-length papers: 5000 words maximum, describing significant completed research.

Poster session abstracts: 1000 words maximum, describing interesting ongoing research.

Demonstration proposals: Videotape and/or description of a live presentation.

Panel discussion proposals: Topic and desired participants.

Submission information

Send all submissions to: Elaine Kant, Dennis O'Neill, Program Chairs, CAIA-88 Schlumberger-Doll Research, Old Quarry Road, Ridgefield, CT 06877-4108

Full-length papers: Submit four copies of the paper (5000 word limit) by September 18, 1987. The first page of the paper should contain the contact information for the author(s) (name, affiliation, address), a 200 word abstract, and a list of appropriate subject categories. Authors are not restricted to only those categories listed above. Accepted papers will be allocated six manuscript pages in the proceedings.

Poster session abstracts: Submit four copies of the abstract (1000 word limit) by November 25, 1987. Indicate all appropriate subject categories. Accepted abstracts will be reprinted and distributed at the conference. In addition, authors of accepted poster session abstracts will be provided with space at the conference to display examples of their work and to discuss their findings with others.

Demonstration proposals: Submit a videotape or a written description of a 10 to 15 minute demonstration by November 25, 1987. The demonstration should be of a particular system or technique that shows the reduction to practice of one of the conference topics.

Panel discussion proposals: Submit a one page description of a topic for panel discussion by November 25, 1987. Indicate the kind of speakers you would like to see participate and whether you are interested in organizing the discussion.

Important dates

Full-length papers due: September 18, 1987
Author notifications mailed: October 16, 1987
Other submissions due: November 25, 1987
Accepted papers to IEEE: December 4, 1987
Conference: March 14-18, 1988, San Diego, CA

Conference committee

General chair
James Miller, MCC
Tutorial chair:
Paul Harmon, Harmon Associates
Program committee chairs:
Elaine Kant, Schlumberger-Doll Research
Dennis O'Neill, Schlumberger-Doll Research
Program committee:
Richard Duda, Syntelligence
Mark Fox, Carnegie-Mellon University
Se June Hong, IBM T.J. Watson Laboratories
Janet Kolodner, Georgia Institute of Technology
Casimir Kulikowski, Rutgers University
John Landry, Distribution Management Systems
Frank Lynch, Digital Equipment Corporation
Nancy Martin, Softpert Systems
Shirley McCarty, The Aerospace Corporation
Ryszard Michalski, University of Illinois
Sanjay Mittal, Xerox Palo Alto Research Center
Fumio Mizoguchi, Tokyo University of Science
Makoto Nagao, Kyoto University
Howard Shrobe, Symbolics
Marilyn Stelzner, Intellicorp
William Swartout, USC-ISI
Richard L. Wexelblat, Philips Laboratories

Additional information

For additional conference information, contact:
CAIA-88
The Computer Society of the IEEE
1730 Massachusetts Avenue, NW
Washington, DC 20036-1903
(202) 371-0101

For information about exhibits, contact:
Richard Greene
Trade Associates, Inc.
12250 Rockville Pike, Suite 200
Rockville, MD 20852
(301) 468-3210

THE COMPUTER SOCIETY OF THE IEEE

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

IEEE

REPORT DOCUMENTATION PAGE

OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Dates attached	

4. TITLE AND SUBTITLE

Titles/Authors - Attached

5. FUNDING NUMBERS

6. AUTHOR(S)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Code FIA - Artificial Intelligence Research Branch
Information Sciences Division

8. PERFORMING ORGANIZATION REPORT NUMBER

Attached

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Nasa/Ames Research Center

Moffett Field, CA. 94035-1000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Available for Public Distribution

Peter Friedland 5/14/92 BRANCH CHIEF

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Abstracts ATTACHED

14. SUBJECT TERMS			15. NUMBER OF PAGES
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT